EECS 373 *Midterm 2*

Fall 2022

Name: ______________________________________ unique name: _______________

Sign the honor pledge code:
"I have neither given nor received unauthorized aid on this examination, nor have I concealed any violations of the Honor Code"

Signature: ___________________________________

>> <u>Do not</u> open the exam until instructed to begin the exam <<

Exam Guidelines:

1. Closed book/notes. Scientific calculators are allowed. No graphing calculators or other electronic devices. Phones must stay in your pockets or bags.
2. There are 15 pages to this exam, including this one
3. You will be provided with the *Fall 2022 Exam 2 Datasheet*
4. You will be provided with the Reference Sheet aka "Appendix 1 & 2"
5. You have 120 minutes for this exam
6. Don't spend too much time on any one problem
7. Show your work and explain what you've done when asked to do so
8. State any assumptions
9. Getting partial credit without showing work will be rare

Question 1: Multiple Choice	16
Question 2: Short Answer	12
Question 3: ADC	12
Question 4: Timers	12
Question 5: Interrupts	8
Question 6: Design Problem - Smart Thermostat	-
Part 1: Initialization	14
Part 2: Helper Functions	10
Part 3: Interrupts	6
Part 4: Main Loop	10
Total	**100**

Question 1: Multiple Choice [2 points each]

For each of the multiple choice questions fill in the circle to indicate the correct answer

Q 1.1

Which of the following is true about ARMv7-M ISA?

- ○ Both big endian and little endian are supported
- ○ The `ANDS` instruction updates APSR, but `AND` does not
- ○ `strh` stores a half-word
- ○ The *Reset_Handler* is the second instruction held in memory
- ○ [✓ All of the above]
- ○ None of the above

Q 1.2

Which of the following statements is ***FALSE*** about ABI compliant functions?

- ○ Values in R4-R11 must be preserved
- ○ Values in R0-R3 are known as “argument” or “scratch” registers
- ○ [✓ The instruction `pop{lr}` can be used to return from a function]
- ○ ABI compliant functions can call other ABI functions
- ○ It is possible to return a 64 bit value even in a 32 bit architecture

Q 1.3

Which of the following can be used to measure the pulse width of a PWM signal?

- ○ Compare timer
- ○ [✓ Capture timer]
- ○ Watchdog timer
- ○ Stopwatch timer
- ○ None of the above

Q 1.4

Given a clock source of 120 MHz and a desired `SysTick` frequency of 500 Hz, what is the desired `SysTick` reload value

- ○ [✓ 239,999]
- ○ 240,000
- ○ 119,999
- ○ 120,000
- ○ None of the above

Q1.5

Which serial protocol is depicted in the following oscilloscope screen shot?

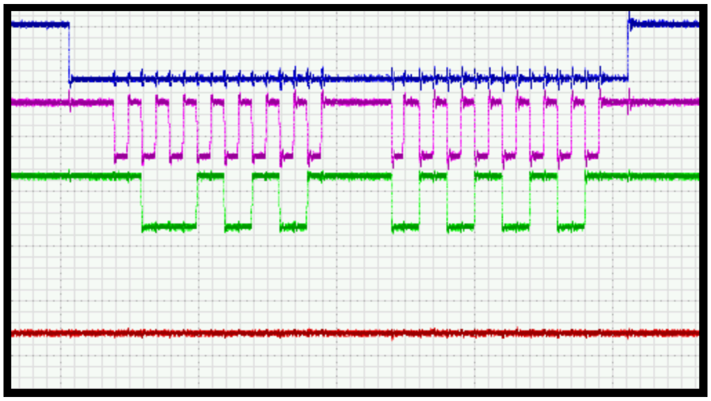
○ UART

[✓ ○ SPI]

○ I2C

○ USB

○ Can not be determined



Q1.6

Sample-and-hold circuits in analog-to digital converters (ADCs) are designed to:

○ stabilize the comparator's threshold voltage during the conversion process

[✓ ○ stabilize the input analog signal during the conversion process]

○ sample and hold the D/A converter staircase waveform during the conversion process

○ sample and hold the output of the binary counter during the conversion process

Q1.7

The number of comparators required for flash type Analog to Digital Converter…

○ triples for each added bit

○ reduce by half for each added bit

[✓ ○ doubles for each added bit]

○ Quadruples for each added bit

Q1.8

A 4-bit R/2R Digital-to-Analog Converter (DAC) has a reference of 5 volts. What is the analog output for the input code 1010?

○ 0.78125 V

○ [✓ 3.125 V]

○ 0.3125 V

○ -3.125 V

Question 2: Short Answers [3 points each]

Q2.1

Why are shift registers an integral part of UART, SPI, and I2C peripheral on STM32 MCUs?

> Shift registers enable the conversion for parallel to serial data formats and vice versa. For example a byte can be written to the output buffer of the serial peripheral which is then sequentially shifted out bit by bit over the serial interface.

Q2.2

On many consumer electronic devices users can perform a “Hard Reset” by holding down the power button for a set amount of time causing the device to reboot. Which is the first interrupt that will run on the device? and where is the address of this interrupt handler stored (be as specific as possible for full credit)?

> On power up the first interrupt that must be run is the reset_handler. The address of the reset_handler is stored in memmory at address 0x04 (i.e the second memory location on the device).

Q2.3 [2 points]

A 44,100Hz timer is used to triggered a DAC which generates a sinusoidal waveform of 293.665 Hz (musical tone D). How many DAC outputs will be produced during one cycle of the sinusoidal waveform?

> 44100/293.665 = 150.17 ≈ 150 DAC outputs/sine wave cycle of tone D

Q2.4

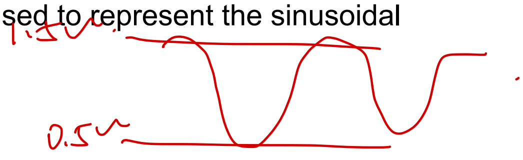
State one reason why SPI has fundamentally higher data throughput than I^2C?

1) SPI uses push-pull CMOS to drive its bus allowing it to drive rising edge transition faster than the I2C resistor can pull the its bus lines from gnd
2) SPI does not have overhead in terms of address bits which reduce
3) Other correct answer accepted

Question 3: ADC

As a new intern in Company X, you have been assigned to work on a large embedded project with a team of 25 engineers, using an MCU based on the Arm v7-M architecture. Your first assignment is to write a function that samples a 100kHz sinusoidal signal, that is 1Vpp (Volt peak to peak), with a 1.0V DC offset. The ADC on the MCU is a 12 Bit SAR ADC, with a sampling rate of 1MHz, a Vref+ of 3.3V and a Vref- of 0V.

Q3.1 What is the number of unique ADC values that can be used to represent the sinusoidal signal? [8 points]



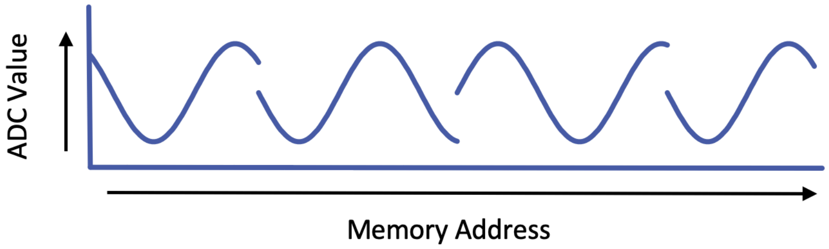
$$ADC\_Output\_Values = (2^{N} - 1) * Vin)/Vref+$$
$$ADC\_top = (2^{12} - 1) * 1.5V)/3.3V = 1861$$
$$ADC\_bottom = (2^{12} - 1) * 0.5V)/3.3V = 620$$

$$\frac{1.5V}{3.3V} * (2^{12} - 1).$$

$$Number\ of\ ADC\ values = ADC\_top - ADC\_bottom$$
$$Number\ of\ ADC\ values = 1861 - 620 = 1241$$
$$Number\ of\ ADC\ values = 1241$$

$$\frac{0.5V}{1.5V} * (2^{12} - 1).$$

Q3.2 Since you are an awesome 373 student you are able to correctly write the function and it works perfectly in isolation on the prototype hardware. You are then asked to merge your code into the main project code base and test out the full system on the prototype hardware. However, when you sample a 100kHz sine wave and read the ADC values out of memory the plot looks like this:



You know that the hardware is working correctly and your code is correct. Give one reason for the discontinuities in the sampled 100kHz sine wave when you merged your code with the main project code. [4 points]

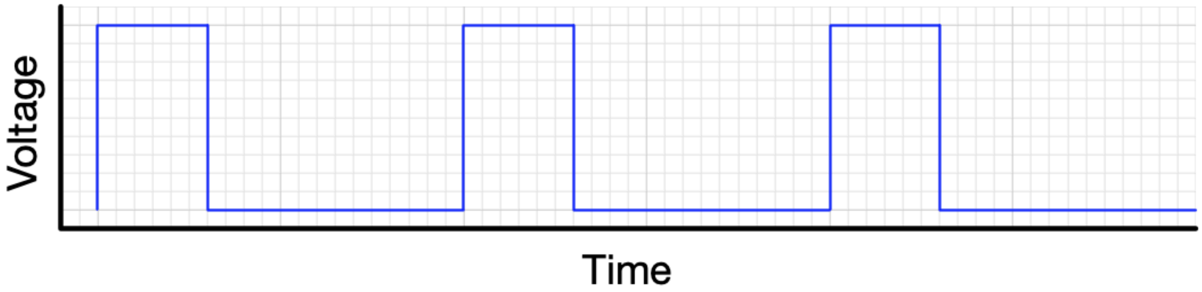
The ADC function that you have written is getting interrupted by higher priority interrupts. This causes the gaps in the ADC reads.

(other sufficiently reasoned answers will be accepted)

Question 4: Timers

Q4.1

A general purpose Timer on the STM32 is driven with a 72MHz clock and has a prescaler value of 5. Using the diagram below determine the ARR and CCR values. Note: The period of the PWM is 5ms and the grid size on the diagram represents 250μS. [6 point]



Period_PWM = (ARR+1) x (PSC+1) / F_CLK
5ms = (ARR+1) x (5+1) / 72MHz
ARR = 59,999 ~ 60,000

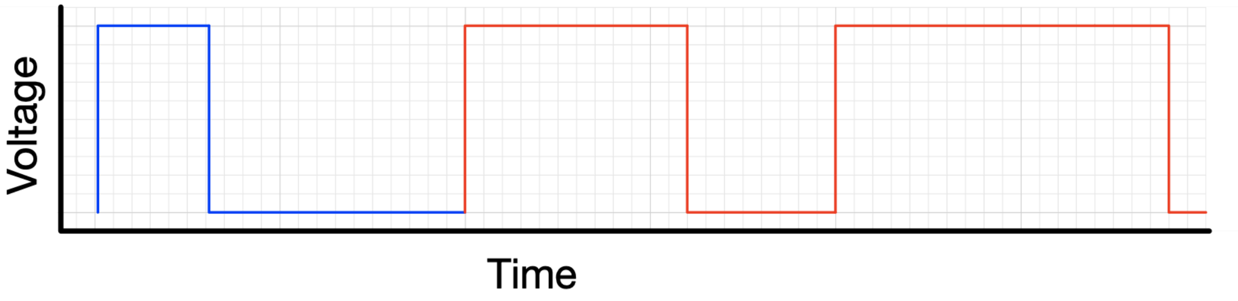
Duty_Cycle = CCR/ARR
6 units/20 units = 0.3 = CCR/60,000
CCR = 18,000

Q4.2

The same Timer as above is reconfigured to run the following interrupt each time the Timer "overflows". Assume the PSC and clock frequency is the same as above. The first cycle of the PWM waveform shown below is the same as the one above. Draw the next two cycles of the PWM waveform resulting from the interrupt. [6 point]

```
void TIM2_Interrupt hander(void){
        temp = TIM2->CCR;
        if (temp > 60000){
                TIM2->CCR = 0;
        }else{
                TIM2->CCR += 18000;
        }
}
```

Answer is the red line of the graph



Duty_Cycle = CCR/ARR

First time interrupt fires the initial value of CCR is 18,000 (as defined in the part 4.1 and shown above in blue) the resulting CCR value is 18,000 +18,000 = 36,000.

Duty_Cycle = 36,000/60,000 = 0.6 = 60%
0.6 * 20 (ticks on the graph per period) = 12 ticks on the graph
Or 3ms

Second time interrupt fires the result CCR value is 18,000 + 36,000 = 54,000

Duty_Cycle = 54,000/60,000 = 0.9 = 90%
0.9 * 20 (ticks on the graph per period) = 18 ticks on the graph
Or 4.5ms

Question 5: Interrupts [8 points]

The following diagram depicts the execution of interrupt events on an STM32. Lightly bolts denote when interrupts are triggered.

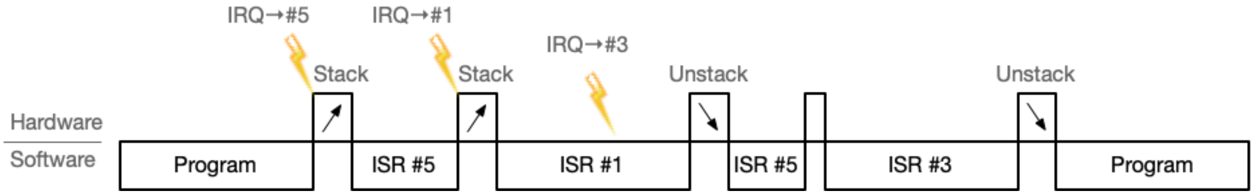
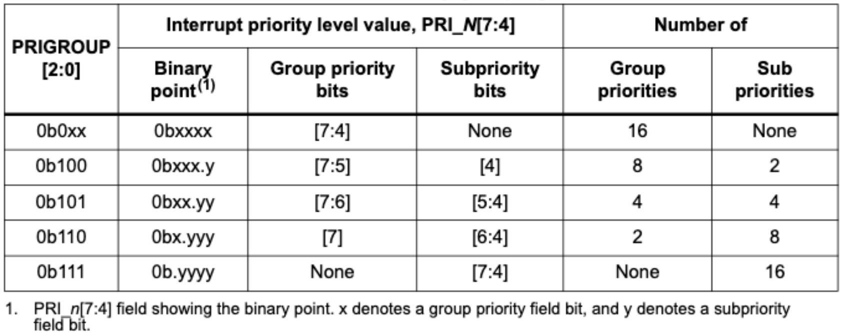


Table 51. Priority grouping



PRIGROUP [2:0]	Interrupt priority level value, PRI_N[7:4]			Number of	
	Binary point[1]	Group priority bits	Subpriority bits	Group priorities	Sub priorities
0b0xx	0bxxxx	[7:4]	None	16	None
0b100	0bxxx.y	[7:5]	[4]	8	2
0b101	0bxx.yy	[7:6]	[5:4]	4	4
0b110	0bx.yyy	[7]	[6:4]	2	8
0b111	0b.yyyy	None	[7:4]	None	16

[1] PRI_n[7:4] field showing the binary point. x denotes a group priority field bit, and y denotes a subpriority field bit.

Q5.1 Using the PRIGROUP 0b110 as defined in the table above, fill out the priority of each of the five interrupts shown in the diagram. [6 points]

ISR #1 priority = _____0000,XXXX__________ (8-digit binary)

ISR #3 priority = _____1001,XXXX__________ (8-digit binary)

ISR #5 priority = _____1000,XXXX__________ (8-digit binary)

Q5.2 What is happening when the program execution transitions from ISR #5 to ISR #3? (short answer) [2 points]

Tail Chaining

Question 6: Design Problem - Smart Thermostat

You have been tasked with designing a home furnace thermostat. It consists of a display and with a button to increase the temperature and a button to decrease the temperature.

Components

Two push buttons normally logic high. When the button is pressed they become logic low. You can assume they are debounced.

1. One MAT32 processor with associated peripherals.
2. One SPI interface TMPF22 digital temperature sensor.
3. One I2C interface DF22 2 line character display.
4. One high voltage furnace control

System Operation

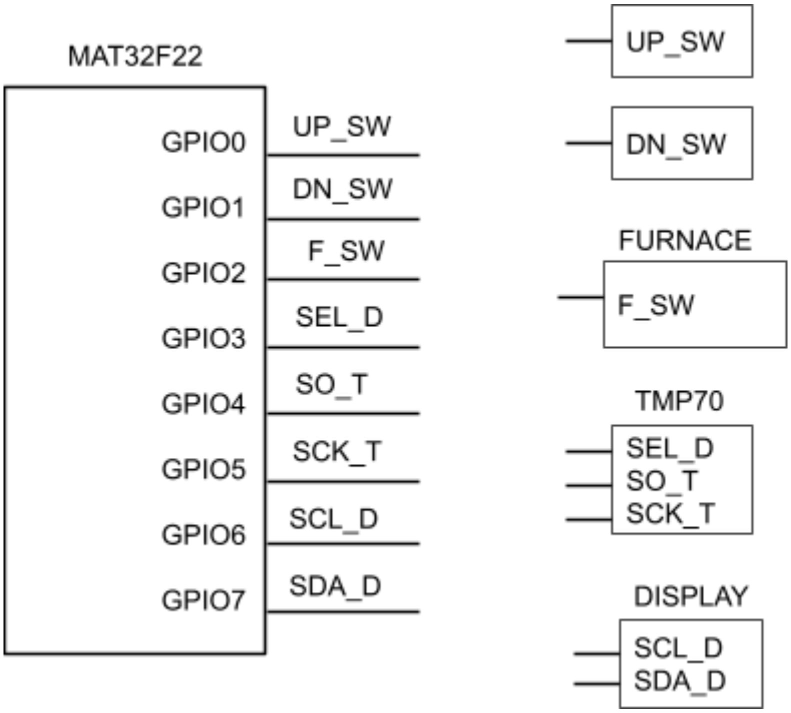
- A timer will periodically drive an interrupt that will read the current temperature and compare it to a set temperature and turn the furnace on or off as needed.
- The set temperature is adjusted with separate up and down button interrupts. The max temperature is 90 degrees Fahrenheit and the minimum temperature is 40 degrees Fahrenheit.
- A main loop will update the display with current set and room temperatures.

Advice

The problem is organized into the components you will need to implement this design.
Read them all over before you begin to see how this organization will support thermostat operation.

Q6.1: GPIO Initialization [14 points]

The GPIO of the MAT32F22 has been connected to the GPIO as follows. Initialize the GPIO to accommodate the device function.



Use the defines provided in the MAT32 GPIO reference, full out the following:

```
MAT_GPIO_Init(GPIO_0, RISING_INT, NAF);
MAT_GPIO_Init(GPIO_1, RISING_INT, NAF);
MAT_GPIO_Init(GPIO_2, OUTPUT, NAF);
MAT_GPIO_Init(GPIO_3, OUTPUT, NAF);
MAT_GPIO_Init(GPIO_4, dc, AF1);
MAT_GPIO_Init(GPIO_5, dc, AF1);
MAT_GPIO_Init(GPIO_6, dc, AF2);
MAT_GPIO_Init(GPIO_7, dc, AF2);
```

Timer Initialization

Provide the C code to initialize one of the timers to interrupt every 10ms. Assume the source clock is 1MHz.

```
base = (int *)0x4000 1000
//set control register to compare mode and enable interrupt
*(base + 0x10) |= 0b0100
//prescale set for 1 multiplier (0)
*(base + 0x4) |= 0
//ARR set for 10ms = 10,000us = 1,000,000us/100
*(base + 0x08) |= 9999;
```

Global Variable and GPIO pin Value Initialization

Provide the C code to initialize any global variables and GPIO outputs. Assume that special functions automatically initialize correctly and GPIO outputs default values are zero.

```
//set_temp is a global variable used to set the control temperature
and is initialized to 70 Fahrenheit.
global char set_temp = 70;// 40 <= set_temp <= 90

//temperature select line must be set high
MAT_GPIO_Write(GPIO_2, 1)
```

Q6.2: Helper Functions [10 Points]

You can assume you have the C libraries listed below. In addition, you will need to write a few helper functions specific to this application.

C Libraries Available.

```
void itoa(int n, char s[])   int strnlen(*str)   char *strcat(char *s1,
const char *s2);
```

Read Temperature Sensor

Provide the C code that reads and returns the temperature. Assume that SPI mode and frame size are already initialized.

```
uint32_t get_temp(void) {
      uint8_t spi_buf[2];

      MAT_GPIO_Write(GPIO_3, 0)
      MAT_SPI_Recieve(*spi_buf, 1)
      MAT_GPIO_Write(GPIO_3, 1)
      return(spi_buf[0])
}
```

Write Display

Provide the C code that updates the display. The function will take 3 arguments: row, col and a pointer to a string. You do not need to do any error checking on the position and string length. You can assume they are reasonable values for the display.

```
void display(uint_8 col, uint_8 row, char *str) {
      char buf[4];

      buf[0]= 0x80;
      buf[1]= col;
      buf[2]= row;

      while(i <= strlen(*str)) {
            buf[2] = ++*str;
            MAT_I2C_Send(0x7, *buf, 3)
      }
}

}
```

Q6.3: Interrupts [6 Points]

Timer Interrupt

Provide the C code that will read the current temperature and turn on the furnace on if the current temperature is **Below or equal** the set temperature point and off otherwise. Assume the set temperature is in the following global variable.

```
void timer_interrupt() {
if (get_temp() >= set_temp)
      MAT_GPIO_Write(GPIO_2, 1);
else
      MAT_GPIO_Write(GPIO_2, 0);
}
```

UP Button Interrupt

Provide the C code to increment the global variable set_temp. Maximum value is 90F.

```
void up_button_interrupt() {
      if(set_temp) <= 90 set_temp++;
}
```

DN Button Interrupt

Provide the C code to decrement the global variable set_temp. Minimum value is 40F.

```
void dn_button_interrupt() {
      if(set_temp) >= 40 set_temp- -;
}
```

Q6.4: Main Loop [10 Points]

The main loop will update the display with the set temperature and current room temperature. Provide the C code using your helper functions to display the set and current room temperature. The set temperature should be prefaced with SET and the room temperature with ROOM. For example,

SET 70
ROOM 67

You don't have to center the characters on the display.

The screen should be cleared before updating it.

Main {

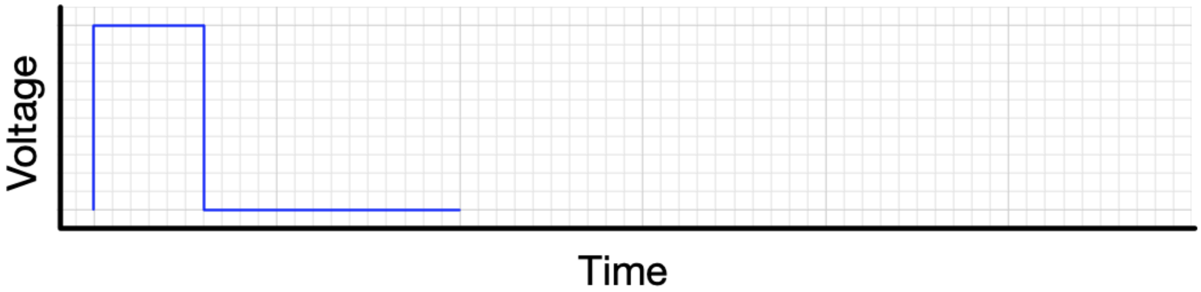
```
//clear screen
char buf[2];
buf[0]= 0x3;
buf[1]= 1;
MAT_I2C_Send(0x7, *buf, 2);

//display set temperature
char str[10] = {0};
itoa(set_temp,str);
strcat(“SET”,str);
void display(0, 0, char *str);


//display room temperature
char str[10] = {0};
itoa(get_temp(),str);
strcat(“SET”,str);
void display(0, 1, char *str);

}
```

Temp Extra


Voltage
Time